# How to implement a modular form?

Martin Raum[a]

[a]*MPI für Mathematik, Vivatsgasse 7, 53111 Bonn, Germany*

**Abstract**

We present a model for Fourier expansions of arbitrary modular forms. This model takes precisions and symmetries of such Fourier expansions into account. The value of this approach is illustrated by studying a series of examples. An implementation of these ideas is provided by the author. We discuss the technical background of this implementation, and we explain how to implement arbitrary Fourier expansions and modular forms. The framework allows us to focus on the considerations of mathematical nature during this procedure. We conclude with a list of currently available implementations and a discussion of possible computational research.

*Keywords:* modular forms, Fourier expansions, computation

## 1. Introduction

The purpose of this paper is twofold: First, we develop a new strategy to compute with Fourier expansions of modular forms. Second, we describe an implementation of this idea in Sage [34].

Explicit computations with modular forms have led to great discovery and they form the basis of many applications. Among them are such important ones like central values of $L$-series or congruences (see [14, 24]). There are many theorems which can be proved by almost purely computational approaches. This comprises the four square theorem, that can be solved with pen and paper (see [6]), as well as the very advanced considerations in [1]. There the proof of a long standing conjecture by Andrew and Alders could be completed. This was mainly done by determining Fourier expansions of a modular form up to sufficient precision. A list of publications that take similar approaches using elliptic modular forms would be very long. Magma [4] and Sage both provide robust implementations of Fourier expansions of such forms.

There is significantly less work that is concerned with higher rank groups. Outstanding contributions in this area have been made by Poor and Yuen [20, 21] and Krieg [8, 11]. Nonetheless, the computations performed so far follow

*Email address:* `MRaum@mpim-bonn.mpg.de` (Martin Raum)

individual implementations that are optimized specific to the type of modular form (degree, subgroup). Such calculations could benefit from a unified framework without sacrificing efficiency.

A first effort to make a unified approach was made by the author and his collaborators in [27]. They suggested an approach to compute with Siegel modular forms and provided an implementation in Sage. In this paper, we suggest a far generalization if this idea, that enables us to treat a multitude of different types of modular forms. Category theory turns out to be the most appropriate language to formulate our model of Fourier expansions. Indeed, all important, technical results that we will make use of have already been proved in this area. This results in a clean and accessible treatment of several issues connected with models of Fourier expansions of modular forms, including precisions and symmetry. Both aspects are directly related to the memory consumption of an implementation. The latter issue can be addressed by carefully extending the notion of $\chi$-invariants with respect to a suitable group $G$. This group $G$ will in fact account for all symmetry that is inherent to the Fourier expansion of a modular form. Precisions will be modeled by morphisms of suitable rings, and the main task consists in relating them cleanly to each other.

This paper is also devoted to an implementation of the above idea that is provided by the author. It was written in Sage, a common and freely available system for mathematical computations, thus permitting the whole community to profit. The development takes place in Purple Sage [33], a library based on Sage that contains code relevant to arithmetic geometry. The idea behind the framework, as we will call it, is that the programmer should need only provide mathematical data without considering questions of purely technical nature. Internally, modular forms will be represented as an abstract element and as a Fourier expansion at the same time. This allows for faster computations and application of specialized routines whenever this is appropriate.

The preliminaries of this paper are contained in Section 2. In Section 3, we explain the mathematical objects which are modeled by our implementation. We will outline several examples and discuss how they are related to Fourier expansions of modular forms. Section 4 contains a description of the concepts that the reader must be familiar with when implementing a new type of modular forms. In Section 5, we illustrate the usage of the framework starting with two examples. We will thereafter proceed to a detailed explanation of the most basic methods that are available for all modular forms. In Section 6, we list types of modular forms that have been implemented based on the framework that we present. Section 7 contains a discussion of the framework's future development and a description of several possible applications.

## 2. Preliminaries

Let $S$ be a monoid, of which the composition is denoted multiplicatively. We fix a ring $R$ with unit and an $R$-left module $A$. In some cases $A$ will carry an $R$-algebra structure. We write $R^\times$ for the group of units in $R$. A left $R$-ideal that is generated by $g_1, \ldots, g_n$ is denoted by ${}_R\langle g_1, \ldots, g_n \rangle$. We write $R\langle x_1, \ldots, x_n \rangle$ for the free rank $n$ module over $R$ with basis $x_1, \dots, x_n$, and we write $R\{x_1, \dots, x_n\}$ for the noncommutative polynomial algebra over $R$ on $n$ variables $x_1, \dots, x_n$. The polynomial algebra with $n$ commuting variables is denoted by $R[x_1, \dots, x_n]$. By $A[S] = \bigoplus_{s \in S} As$ we denote the direct sum of copies of $A$ indexed by $S$. If $A$ is an algebra this is the monoid algebra of $S$ over $A$. It is an $R[S]$-left module in all cases. In addition to the usual notion of finite rank modules, we use the following notion: A module that is isomorphic to a sum of finitely many copies of $A$ is said to have finite rank over $A$.

We write $\mathrm{M}_{n,m}(R)$ for the set of $n \times m$ matrices with entries in $R$. We abbreviate $\mathrm{M}_{n,n}(R)$ by $\mathrm{M}_n(R)$. The submodule of symmetric matrices is denoted by $\mathrm{M}_n^T(R)$. The general linear group is denoted by $\mathrm{GL}_n(R) \subseteq \mathrm{M}_n(R)$, and the symplectic group is denoted by $\mathrm{Sp}_n(R) \subseteq \mathrm{M}_{2n}(R)$. The transpose of $u \in \mathrm{M}_n(R)$ is denoted by ${}^{\mathrm{T}}u$. We set $m[u] := {}^{\mathrm{T}}umu$, if $m, u \in \mathrm{M}_n(R)$.

Let $G$ be a group acting on $R$, and suppose that $\chi : G \to R^\times$ is a character. We denote the $\chi$-invariants in $R$ by

$$R^G_\chi := \{r \in R : gr = \chi(g)r \text{ for all } g \in G\}.$$

For a group $\mathbb{G}\mathrm{r}$ a $\mathbb{G}\mathrm{r}$-grading of $R\{x_1, \dots, x_n\}$ is a partially defined homomorphism $\mathrm{gr} : R\{x_1, \dots, x_n\} \to \mathbb{G}\mathrm{r}$, such that all $\mathrm{gr}^{-1}(\{g\})$ for $g \in \mathbb{G}\mathrm{r}$ are $R$-left submodules of $A\{x_1, \dots, x_n\}$ and $A\{x_1, \dots, x_n\} = \bigoplus_{g \in \mathbb{G}\mathrm{r}} \mathrm{gr}^{-1}(\{g\})$. A grading of $R\langle x_1, \dots, x_n\rangle$ is defined analogously, where gr is a map of sets. In this case it suffices to assume that $\mathbb{G}\mathrm{r}$ is a set.

The reader of Section 3 is required to be familiar with basic constructions in category theory. In all other sections we avoid this terminology, specializing the results from Section 3 to a language that readers with a background in modular forms will feel more comfortable with. For an introduction to category theory we refer the reader to [18].

For any category $\underline{\Lambda}$ we denote by $\mathrm{Ob}(\underline{\Lambda})$ the objects of $\underline{\Lambda}$, and we denote by $\underline{\Lambda}(a, b)$ the arrows between $a, b \in \mathrm{Ob}(\underline{\Lambda})$. Most of the time, we say morphism for an arrow in categories that we deal with. We agree on denoting all categories by underlined letters or words and all functors by capital script letters. A category $\underline{\Lambda}$ is said to be a net if $\#\underline{\Lambda}(a, b) \in \{0, 1\}$ for all $a, b \in \mathrm{Ob}(\underline{\Lambda})$ and if we can always find $c \in \mathrm{Ob}(\underline{\Lambda})$ such that $\#\underline{\Lambda}(a, c) = \#\underline{\Lambda}(b, c) = 1$. Notice that any functor with codomain a net is uniquely defined by the associated assignment of objects. We denote the evaluation of a functor $\mathscr{F}$ on objects $\lambda \in \mathrm{Ob}(\underline{\Lambda})$ and on morphisms $m \in \underline{\Lambda}(\lambda, \lambda')$ by $\mathscr{F}(\lambda)$ and $\mathscr{F}(m)$. The limit of a functor $\mathscr{F}$ is denoted by $\lim \mathscr{F}$. For a definition of limits the reader is referred to [18, Sec III.4].

The small category with the multiplicatively closed subsets of $S$ as objects is denoted by $\underline{\mathrm{Mult}}(S)$. Given $a, b \in \mathrm{Ob}(\underline{\mathrm{Mult}}(S))$ the set of morphisms $\underline{\mathrm{Mult}}(S)(a, b)$ contains a unique element if $a \subseteq b$, and it is empty, otherwise. In particular, $\underline{\mathrm{Mult}}(S)$ is a net. We now introduce several full subcategories of $\underline{\mathrm{Mult}}(S)$.

A set $S' \subseteq S$ is called absorbing if $s's, ss' \in S'$ for any $s \in S$ and any $s' \in S'$. We call a subset $S' \subseteq S$ cofinite, if $S \smallsetminus S'$ is finite. Let $\underline{\mathrm{Abs}}(S)$ be the category of subsets of $S$ that are absorbing. The category of cofinite, absorbing subsets of $S$ is denoted by $\underline{\mathrm{Abs}}_{\mathrm{cofinite}}(S)$.

Given a group $G$ that acts on $S$ we denote by $\underline{\mathrm{Abs}}^G(S)$ the category of absorbing subsets of $S$ that are $G$-invariant. We write $\underline{\mathrm{Abs}}_{G-\mathrm{cofinite}}(S)$ for the category of all $G$-cofinite subsets of $S$. A set $S' \subseteq S$ is called $G$-cofinite if $S \setminus S'$ is the union of finitely many $G$-orbits. Notice that this implies $G$-invariance.

The category $\underline{\mathrm{Mod}}(R)$ is the category of $R$-left modules. Its subcategory $\underline{\mathrm{SMod}}(A, R)$ is the category of $R$-left submodules of $A$ with arrows given by inclusions.

There is a covariant functor $\mathscr{S}(A, S) : \underline{\mathrm{Abs}}(S) \to \underline{\mathrm{SMod}}(A[S], R)$ that assigns ${}_{A[S]}\langle As \,:\, s \in a\rangle$ to $a \in \mathrm{Ob}(\underline{\mathrm{Abs}}(S))$. To every arrow we assign the obvious inclusion. There is also a functor $\mathscr{Q}(A[S], S) : \underline{\mathrm{SMod}}(A[S], R) \to \underline{\mathrm{Mod}}(R)$. On objects it is defined by $A[S] \supseteq I \mapsto A[S]/I$. Morphisms are mapped to natural epimorphisms between quotients of $A[S]$.

In Section 5, the reader is required to know basis commands in Sage [34]. This system for computer supported mathematics is based on Python [36], a widely spread script language that is easy to learn. For a basic introduction the reader is referred to [35]. In [39] the reader will find a very gentle introduction to many functionalities of Sage. A basic reference for programming with Python is provided in [37]. We emphasize that the reader only interested in using implementations that are already provided will not need read this.

## 3. The underlying construction

This section contains a description of how Fourier expansions and their connection to modular forms can be modeled for efficient use in implementations.

The basic tool for modeling Fourier expansions will be monoid power series, that we introduce in Section 3.1. The Fourier expansion of the major part of modular forms has symmetries that can be described by the action of a group on both its Fourier indices and its coefficients. Equivariant monoid power series are best suited to describe these symmetries. We define modules of equivariant monoid power series in Section 3.2. We also discuss under which assumptions these modules are rings.

Fourier expansions are intimately connected to modular forms, but they do not capture all aspect an implementation should cover. To describe all important properties we introduce a further ring (or module), that we call a ring (or module) of graded expansions. Even though, mathematically speaking, it is only a homomorphism; the reader should carefully read Section 3.3. In that section, we discuss the mathematical aspects of what the user of any implementation building on the framework that we describe in Section 4 will deal with.

Throughout this section we fix a monoid $S$ and a ring $R$ with units, that is not necessarily commutative. We write $A$ for a fixed $R$-left module. In many cases $A$ will carry an algebra structure, and we will comment on theses cases separately, whenever it seems appropriate.

### *3.1. Monoid power series*

We fix a small category $\underline{\Lambda}$, and we assume that it is a net. There is a one-to-one correspondence between functors $\mathscr{F} : \underline{\Lambda} \to \underline{\mathrm{Abs}}(S)$ and filtrations of $S$ that are indexed by $\mathrm{Ob}(\underline{\Lambda})$. The functor $\mathscr{F}$ ultimately provides a way to model truncation of Fourier expansions. The fact that the codomain is the category of absorbing subsets of $S$ guarantees that truncation and multiplication of monoid power series are compatible. The functor

$$\mathscr{F}^{\mathrm{Quot}(A)} := \mathscr{Q}(A[S], S) \circ \mathscr{S}(A, S) \circ \mathscr{F}$$

is covariant.

**Definition 3.1** *Let $\mathscr{F} : \underline{\Lambda} \to \underline{\mathrm{Abs}}(S)$ be a covariant functor. The* ring (or module) of monoid power series *over $A$ with respect to $\mathscr{F}$ is*

$$A[\![\mathscr{F}]\!] := \lim \mathscr{F}^{\mathrm{Quot}(A)}.$$

**Remarks 3.2**

1. *If the codomain of $\mathscr{F}$ is the category of cofinite subsets of $S$ and if $A$ is an algebra, the module $A[\![\mathscr{F}]\!]$ will be an algebra.*

2. *Notice that $A[\![\mathscr{F}]\!]$ is not a useful object in all cases. The most striking example is $\mathscr{F} : \lambda \mapsto S$ for all $\lambda \in \mathrm{Ob}(\underline{\Lambda})$. Then $A[\![\mathscr{F}]\!]$ is the ring with one element.*

3. *Because $\mathscr{F}$ maps into the category of absorbing subsets of $S$, we may write $\sum_{s \in S} a_s s$ with $a_s \in A$ for elements in $A[\![\mathscr{F}]\!]$.*

**Example 3.3 (Multivariat power series)** *Set $S = (\mathbb{N}^m, +)$, and the objects in $\underline{\Lambda}$ will be $\mathrm{Ob}(\underline{\Lambda}) = \mathbb{N} \cup \{\infty\}$. The category $\underline{\Lambda}$ is equipped with arrows $a \to b$ that correspond to the relations $a \le b$. The functor $\mathscr{F}$ is defined by means of $a \mapsto \{(n_1, \dots, n_m) : \sum n_i \ge a\}$. Since $\underline{\mathrm{Abs}}(S)$ is a net, it is uniquely defined by this assignment. If $A$ is an algebra, we obtain the classical power series ring in multiple variables $A[\![\mathscr{F}]\!] = A[\![x_1, \dots, x_m]\!]$.*

**Example 3.4 (Siegel modular forms)** *Set $R = A = \mathbb{Q}$ and let $S \subseteq \mathrm{M}_n^T(\mathbb{Q})$ be the set of all semi-positive definite, even, symmetric $n \times n$ matrices. We denote the trace of $s \in S$ by $\mathrm{tr}(s)$. Set $\mathrm{Ob}(\underline{\Lambda}) = \mathbb{N} \cup \{\infty\}$ as above. We define $\mathscr{F}$ by $a \mapsto \{s \in S : \mathrm{tr}(s) \ge a\}$. The ring $\mathbb{Q}[\![\mathscr{F}]\!]$ models the ring of Fourier expansions of Siegel modular forms of degree $n$. More precisely, let*

$$A_\Gamma(\mathbb{Q}) := \bigoplus_{k \in \mathbb{Z}} [\Gamma, \det{}^k]_{\mathbb{Q}}$$

*be the graded ring of Siegel modular forms for a finite index subgroup $\Gamma \subseteq \mathrm{Sp}_n(\mathbb{Z})$ that are defined over $\mathbb{Q}$. Assume that $\mathrm{M}_n^T(\mathbb{Z}) \subseteq \Gamma$ is realized as matrices $\left(\begin{smallmatrix} I_n & T \\ & I_n \end{smallmatrix}\right)$. Then there is an injective homomorphism $A_\Gamma(\mathbb{Q}) \hookrightarrow \mathbb{Q}[\![\mathscr{F}]\!]$ given by the Fourier expansion of a modular form.*

We will now show that $A[\![\mathscr{F}]\!]$ only depends on $\mathscr{F}$ up to a certain equivalence, if $\mathscr{F} : \underline{\Lambda} \to \underline{\mathrm{Abs}}_{\mathrm{cofinite}}(S)$.

**Proposition 3.5** *Assume that two functors*

$$\mathscr{F} : \underline{\Lambda} \to \underline{\mathrm{Abs}}_{\mathrm{cofinite}}(S) \quad \textit{and} \quad \mathscr{F}' : \underline{\Lambda}' \to \underline{\mathrm{Abs}}_{\mathrm{cofinite}}(S)$$

*satisfy* $\lim \mathscr{F} = \lim \mathscr{F}'$. *Then* $A[\![\mathscr{F}]\!] \cong A[\![\mathscr{F}']\!]$.

**Proof** It suffices to construct a homomorphism $A[\![\mathscr{F}]\!] \to A[\![\mathscr{F}']\!]$. It will be clear that the same construction with $\mathscr{F}$ and $\mathscr{F}'$ interchanged yields its inverse.

Fix $\lambda' \in \mathrm{Ob}(\underline{\Lambda})$. Since $\mathscr{F}'(\lambda')$ is cofinite and $\mathscr{F}'(\lambda') \supseteq \lim \mathscr{F} = \bigcap_\lambda \mathscr{F}(\lambda)$, we can find $\lambda$ such that $\mathscr{F}(\lambda) \subseteq \mathscr{F}'(\lambda')$. Consequently, for all $\mu \to \lambda$ there is a well-defined epimorphism

$$A[\![\mathscr{F}]\!] \to \mathscr{F}^{\mathrm{Quot}(A)}(\mu) \to \mathscr{F}'^{\mathrm{Quot}(A)}(\lambda').$$

These morphisms are compatible with the structure of $\underline{\Lambda}$, and by the universal property of $A[\![\mathscr{F}']\!]$, we obtain a morphism $A[\![\mathscr{F}]\!] \to A[\![\mathscr{F}']\!]$. This completes the proof. □

The assumptions on $\mathscr{F}$ are chosen such that in addition to $A[\![\mathscr{F}]\!]$ we can obtain morphisms to modules that have finite rank over $A$. We can use these to store information about $A[\![\mathscr{F}]\!]$ and its elements.

**Definition 3.6** *Suppose that* $\mathscr{F} : \underline{\Lambda} \to \underline{\mathrm{Abs}}_{\mathrm{cofinite}}(S)$ *is a covariant functor. The* ring (or module) of monoid power series with epimorphisms *over $A$ attached to $\mathscr{F}$ is a pair* $\big(A[\![\mathscr{F}]\!], (\epsilon_\lambda)_{\lambda \in \mathrm{Ob}(\underline{\Lambda})}\big)$ *of a ring (or module) of monoid power series and the natural epimorphisms* $A[\![\mathscr{F}]\!] \to \mathscr{F}^{\mathrm{Quot}(A)}(\lambda)$ *indexed by* $\mathrm{Ob}(\underline{\Lambda})$.

**Remark 3.7** *As we will see below, for typical choices of $\mathscr{F}$, the algebra (or module) $A[\![\mathscr{F}]\!]$ has infinite rank over $A$. Since* $\mathscr{F} : \underline{\Lambda} \to \underline{\mathrm{Abs}}_{\mathrm{cofinite}}(S)$, *the algebras (or modules)* $\mathscr{F}^{\mathrm{Quot}(S)}(\lambda)$ *have finite rank over $A$. We may use them to store approximations of elements of $A[\![\mathscr{F}]\!]$.*

**Example 3.8 (Multivariate power series)** *Define $S$ and $\underline{\Lambda}$ be as in Example 3.3. We consider two functors $\mathscr{F}$ and $\mathscr{F}'$. The former is defined by the assignment* $a \mapsto \{(n_1, \ldots, n_m) : \sum n_i \ge a\}$, *and the latter is defined by* $a \mapsto \{(n_1, \ldots, n_m) : n_i \ge a \text{ for some } i\}$. *Using Proposition 3.5, we see that* $A[\![\mathscr{F}]\!] \cong A[\![\mathscr{F}']\!]$. *But the attached rings (or modules) of monoid power series with epimorphisms differ. Using the morphisms $\epsilon_\lambda$ associated to $\mathscr{F}$ it will be more expensive to store an exact version of* $x_1^{50} + x_1^{50} x_2^{50}$ *than to store an exact version of* $x_1^{80} + x_1^{20} x_2^{60}$. *The former polynomial is more efficiently stored applying the epimorphisms associated to $\mathscr{F}'$.*

The next proposition shows that our new construction only yields additional structure if $\underline{\Lambda}$ has no initial object.

**Proposition 3.9** *If $\underline{\Lambda}$ has an initial object* $\lambda_{\mathrm{in}}$, *then* $A[\![\mathscr{F}]\!] \cong \mathscr{F}^{\mathrm{Quot}(A)}(\lambda_{\mathrm{in}})$.

**Proof** This follows from the universal properties of lim. □

**Corollary 3.10** *Suppose* $\lim \mathscr{F}$ *is cofinite. Then* $A[\![\mathscr{F}]\!] \cong \mathscr{F}^{\mathrm{Quot}(A)}(\lambda)$ *for a suitable* $\lambda \in \mathrm{Ob}(\underline{\Lambda})$.

**Proof** Choose $\lambda$ such that $\mathscr{F}(\lambda) = \lim \mathscr{F}$. □

*3.2. Equivariant monoid power series*

Example 3.4 illustrates that monoid power series are not best suited to store Fourier expansions of modular forms: The group $\mathrm{GL}_n(\mathbb{Z}) \hookrightarrow \mathrm{Sp}_n(\mathbb{Z})$ acts on the Fourier expansion of Siegel modular forms for the full modular group $\mathrm{Sp}_n(\mathbb{Z})$. This results in the symmetry $a_s = \pm a_{s'}$ if $s' = s[u]$ for some $u \in \mathrm{GL}_n(\mathbb{Z})$.

For the rest of this section we fix a group $G$ acting on $S$ and $A$ respecting the structure of both, and we fix a character $\chi : G \to R^\times$. It is immediate that $A[\![\mathscr{F}]\!]$ admits a $G$-action that respects the $R$-module structure, whenever $\mathscr{F} : \underline{\Lambda} \to \underline{\mathrm{Abs}}(S)_{G\text{−cofinite}}$.

**Definition 3.11** *Assume that* $\mathscr{F} : \underline{\Lambda} \to \underline{\mathrm{Abs}}(S)_{G\text{−cofinite}}$ *is a covariant functor. The* module of equivariant monoid power series *attached to* $\mathscr{F}$*,* $G$ *and* $\chi$ *is*

$$A[\![\mathscr{F}]\!]^G_\chi := \big\{ r = \sum a_s s \in A[\![\mathscr{F}]\!] \,:\, \chi(g) \sum (g a_s)(gs) = r \textit{ for all } g \in G \big\}.$$

Loosely speaking, $A[\![\mathscr{F}]\!]^G_\chi$ is the module of $\chi$-invariants in $A[\![\mathscr{F}]\!]$ (see [32] for a definition of $\chi$-invariants). If $\chi = \mathbb{1}$ is the trivial character we write $A[\![\mathscr{F}]\!]^G$ for $A[\![\mathscr{F}]\!]^G_\chi$.

For any $\mathscr{F} : \underline{\Lambda} \to \underline{\mathrm{Abs}}(S)$ we may define $\mathscr{F}^G : \underline{\Lambda} \to \underline{\mathrm{Abs}}^G(S)$ with assignment $\mathrm{Ob}(\underline{\Lambda}) \ni \lambda \mapsto S \smallsetminus \big(G(S \smallsetminus \mathscr{F}(\lambda))\big)$. Notice that $\mathscr{F}^G = \mathscr{F}$ if $\mathscr{F} : \underline{\Lambda} \to \underline{\mathrm{Abs}}^G(S)$. By abuse of notation we write $A[\![\mathscr{F}]\!]^G_\chi$ for $A[\![\mathscr{F}^G]\!]^G_\chi$.

In analogy to what we have done in Section 3.1, we define equivariant monoid power series with epimorphisms attached to them.

**Definition 3.12** *Assume that* $\mathscr{F} : \underline{\Lambda} \to \underline{\mathrm{Abs}}(S)_{G\text{−cofinite}}$ *is a covariant functor. By a* module of equivariant monoid power series with epimorphisms *we mean a pair* $\big(A[\![\mathscr{F}]\!]^G_\chi, (\epsilon_\lambda)_{\lambda \in \mathrm{Ob}(\underline{\Lambda})}\big)$*, where* $\epsilon_\lambda$ *is the family of natural epimorphisms* $A[\![\mathscr{F}]\!]^G_\chi \to \mathscr{F}^{\mathrm{Quot}(A)}(\lambda)$ *indexed by* $\mathrm{Ob}(\underline{\Lambda})$*.*

**Remarks 3.13**

1. *Notice that the modules of* $\chi$*-invariants* $\mathscr{F}^{\mathrm{Quot}(A)}(\lambda)^G_\chi$ *have finite rank over* $A$*. We hence may use the epimorphisms* $\epsilon_\lambda$ *to store information about elements of* $A[\![\mathscr{F}]\!]^G_\chi$*.*

2. *The analogs of Proposition 3.5 and 3.9 and Corollary 3.10 hold true for modules of equivariant monoid power series.*

3. *Passing from* $\mathscr{F}$ *to* $\mathscr{F}^G$ *can be done transparently by the implementation that we discuss in the next section. There is no need to consider* $G$*-invariance directly as a user of the framework.*

In this section we consider functors $\mathscr{F} : \underline{\Lambda} \to \underline{\mathrm{Abs}}(S)_{G\text{−cofinite}}$. For that reason, $A[\![\mathscr{F}]\!]$ does not always carry an algebra structure, even if $A$ is an algebra. We investigate under which circumstances a module of equivariant monoid power series carries a natural algebra structure.

**Definition 3.14** *A monoid $S$ is called decomposition finite, if for all $s \in S$, there are only finitely many pairwise distinct pairs $(s_1, s_2) \in S \times S$ that satisfy $s_1 s_2 = s$.*

**Proposition 3.15** *Suppose that $A$ is an $R$-left algebra, and suppose that $S$ is decomposition finite.*

1. *Then $A[\![\mathscr{F}]\!]^G$ is an algebra, and*
2. *if $A'$ is an $A$-left module, $A'[\![\mathscr{F}]\!]^G$ is an $A[\![\mathscr{F}]\!]^G$-module.*

**Proof** The multiplication in $A[\![\mathscr{F}]\!]$ is given by

$$\Big(\sum a_s s\Big)\Big(\sum b_s s\Big) = \sum\Big(\sum_{s_1 s_2 = s} a_{s_1} b_{s_2}\Big)s.$$

The module structure on $A'[\![\mathscr{F}]\!]$ can be defined analogously. □

Let $C$ be a monoid. For $\hat{\chi} : C \to \mathrm{Hom}(G, R^\times)$ we can consider the direct sum of modules $A[\![\mathscr{F}]\!]^G_{\hat{\chi}} := \bigoplus_{c \in C} A[\![\mathscr{F}]\!]^G_{\hat{\chi}(c)}$. It will cause no confusion if we also call this module a *module of equivariant monoid power series*. The next proposition tells us under which conditions it carries a multiplicative structure.

**Proposition 3.16** *Suppose that $A$ is an $R$-left algebra, and suppose that $S$ is decomposition finite.*

1. *Then $A[\![\mathscr{F}]\!]^G_{\hat{\chi}}$ is an algebra, and*
2. *if $A'$ is an $A$-left module, $A'[\![\mathscr{F}]\!]^G_{\hat{\chi}}$ is an $A[\![\mathscr{F}]\!]^G_{\hat{\chi}}$-module.*

**Proof** The multiplication can be defined as in the proof of Proposition 3.15. □

We now give several examples to illustrate how to apply our construction to modular forms.

**Example 3.17 (Siegel modular forms with characters)** *Set $R = A = \overline{\mathbb{Q}}$, and we define $S$ as in Example 3.4. By the block diagonal embedding of $\mathrm{GL}_n(\mathbb{Z})$ into $\mathrm{Sp}_n(\mathbb{Z})$ we mean the morphism $u \mapsto \mathrm{diag}(u, {}^{\mathrm{T}}u^{-1})$. We denote the image of this embedding by $\mathrm{GL}_n^{\mathrm{d}}(\mathbb{Z})$. We fix a congruence subgroup $\Gamma \subseteq \mathrm{Sp}_n(\mathbb{Z})$ subject to the same assumption as in Example 3.4. Characters (with values in $\overline{\mathbb{Q}}^\times$) of $\Gamma$ give rise to characters of $\mathrm{GL}_n^{\mathrm{d}}(\mathbb{Z}) \cap \Gamma$. The set $C$ of characters of $\mathrm{GL}_n^{\mathrm{d}}(\mathbb{Z}) \cap \Gamma$ is finite.*

*Set $S' := S \times \big(\mathrm{GL}_n^{\mathrm{d}}(\mathbb{Z}) / (\Gamma \cap \mathrm{GL}_n^{\mathrm{d}}(\mathbb{Z}))\big)$. The functor*

$$\mathscr{E} : \underline{\mathrm{Mult}}(S) \to \underline{\mathrm{Mult}}(S'),\ \mathrm{Ob}(\underline{\mathrm{Mult}}(S)) \ni a \mapsto \{(s, g) : s \in a\}$$

*is faithful. Using the functor $\mathscr{F}$ from Example 3.4, we set $\mathscr{F}' = \mathscr{E} \circ \mathscr{F}$. In the sense of Example 3.4 the elements of $\overline{\mathbb{Q}}[\![\mathscr{F}']\!]_C^{\mathrm{GL}_n^{\mathrm{d}}(\mathbb{Z})}$ model the Fourier expansions of Siegel modular forms for $\Gamma$ with character.*

*The same construction applies to paramodular forms, that Siegel modular forms fom a particular case of.*

**Example 3.18 (Vector-valued elliptic modular forms)** *Vector-valued elliptic modular forms can be considered maps $\mathbb{H}_1 \to \mathbb{C}[L^\vee/L]$ for a given integral, even lattice $L$. Here $L^\vee$ is the dual of $L$. The action of $\mathrm{SL}_2(\mathbb{Z})$ on the right hand side is given by the Weil representation, that factors over a suitable congruence subgroup $\Gamma$ [3]. In particular, we can consider any such modular form a tuple of scalar-valued elliptic modular forms for this group $\Gamma$. The level $N$ of $\Gamma$ is at most twice the level of $L$. Consequently, we choose $S = (\frac{1}{2N}\mathbb{N}, +)$ and $R = \mathbb{Q}$, $A = \mathbb{Q}[L^\vee/L]$. In analogy to Example 3.3, we choose $\mathscr{F}$ the functor from $\underline{\Lambda}$ with $\mathrm{Ob}(\underline{\Lambda}) = \mathbb{N} \cup \{\infty\}$ to $S$. Then $A[\![\mathscr{F}]\!]$ models the module of Fourier expansions of vector-valued elliptic modular forms.*

*The action of scalar-valued modular forms on this module can be modeled by repeating the construction with $L = (1)$ (that is not even but still works). The resulting ring $\mathbb{Q}[\![\mathscr{F}]\!]$ acts on $A[\![\mathscr{F}]\!]$, since $\mathbb{Q}$ acts on $\mathbb{Q}[L^\vee/L]$.*

**Example 3.19 (Vector-valued Siegel modular forms)** *We adopt the notation and the definitions from Example 3.17, and we fix $n = 2$. We set $R = \mathbb{Q}$ and $A = \mathbb{Q}[x,y]_l$, the space of homogeneous polynomials in $x$ and $y$ of degree $l$. The action of $\mathrm{GL}_2(\mathbb{Z})$ on $A$ is given by $A \ni (g,p) \mapsto p \circ g$. Notice that $A$ is isomorphic to the symmetric power $\mathrm{Sym}^l(\mathbb{Q}^2)$ as a $\mathrm{GL}_2(\mathbb{Q})$-module. The Fourier expansions of vector-valued Siegel modular forms [6] for $\mathrm{Sp}_2(\mathbb{Z})$ are modeled by elements of $A[\![\mathscr{F}]\!]^{\mathrm{GL}_2^{\mathrm{d}}(\mathbb{Z})}$.*

**Remark 3.20** *Obviously, Example 3.17 and 3.19 can be combined. The resulting construction makes use of all important features of Definition 3.12.*

Finally, we consider the following generalization of modules of equivariant monoid power series: We drop the assumption that the action of $G$ respects the monoid structure of $S$. Obviously, in this case $A[\![\mathscr{F}]\!]^G$ is not an algebra. In particular, Proposition 3.15 does not generalize to this case.

We assume that $G = G' \times H$ for two groups $G'$ and $H$, and we fix $S' \subseteq S$. We suppose that $H$ acts trivially on $S'$ and that $(g,h)(s's) = (gs')((g,h)s)$ for all $(g,h) \in G$, $s' \in S'$ and $s \in S$. We consider a functor $\mathscr{F}' : \underline{\Lambda} \to \underline{\mathrm{Mult}}(S')$. The $R$-module $A[\![\mathscr{F}]\!]^G$ will also carry an $R[\![\mathscr{F}']\!]^{G'}$-algebra structure. The next example illustrates this generalization, that is currently only partially covered by the author's framework.

**Example 3.21 (Jacobi forms)** *We consider Jacobi forms of index $m > 0$. Let $R = A = \mathbb{Q}$ and $S = \{(n,r) : 4nm \ge r^2\} \subseteq (\mathbb{Z}^2, +)$. The full Jacobi group, defined as $\Gamma^{\mathrm{J}} := \mathrm{SL}_2(\mathbb{Z}) \ltimes \mathbb{Z}^2$, contains $\mathbb{Z} \subseteq \mathbb{Z}^2$, that acts on $S$ via the map $S \times \mathbb{Z}^2 \to S\,(1,(n,r)) \mapsto (n - r + m, r - 2m)$. We may choose $H = \mathbb{Z}$, $G' = \{1\}$ and $S' = (\mathbb{N}, +)$. With $\mathrm{Ob}(\underline{\Lambda}) = \mathbb{N} \cup \{\infty\}$ and $\mathscr{F}$ given on objects by the assignment $a \mapsto \{(n,r) : n \ge a\}$ and $\mathscr{F}'$ by $a \mapsto \{n : n \ge a\}$ we obtain the Fourier expansions $\mathbb{Q}[\![\mathscr{F}']\!]^{G'} = \mathbb{Q}[\![\mathscr{F}']\!]$ of elliptic modular forms, that act on the Fourier expansions $\mathbb{Q}[\![\mathscr{F}]\!]^G$ of Jacobi forms.*

We can avoid using the generalized module of equivariant monoid power series as follows:

**Example 3.22 (Jacobi forms; an alternative construction)** *The full Jacobi group can be embedded into* $\mathrm{Sp}_2(\mathbb{Z})$*. We modify the construction given in Example 3.4. The group* $G = \mathbb{Z} \subset \mathrm{GL}_2(\mathbb{Z})$ *realized as upper triangular, unipotent matrices embeds into* $\mathrm{GL}_2^{\mathrm{d}}(\mathbb{Z})$*. The definition of* $\mathrm{GL}_2^{\mathrm{d}}(\mathbb{Z})$ *is taken from Example 3.17. The functor* $\mathscr{F}$ *from Example 3.4 now can be used to obtain an alternative ring* $\mathbb{Q}[\![\mathscr{F}]\!]^G$ *of Fourier expansions of Jacobi forms with an arbitrary index* $m > 0$*. The Fourier expansion of a Jacobi form of index* $m$ *is indexed by* $\left\{\left(\begin{smallmatrix} n & r/2 \\ r/2 & m \end{smallmatrix}\right) : n, r \in \mathbb{Z}\right\} \subseteq \mathrm{M}_2^T(\mathbb{Z})$.

## *3.3. Graded expansions*

We want to treat modular forms as abstract elements. This usually happens for one of two reasons. Our ultimate goal might be computing the Fourier expansion of modular forms. Calculating abstractly with modular forms and obtaining their Fourier expansions by means of specialized methods yields a much better performance, although it depends on a detailed understanding of special subspaces. Examples are provided by Maaß lifts in the case of Siegel modular forms. In various other cases, we know the algebraic structure of a module or ring of modular forms, even though we cannot compute the Fourier coefficients of arbitrary elements without expressing them in terms of these generators. Orthogonal modular forms for signature $(2, n)$, $n \geq 4$ provide typical examples of this situation. If we understand modular symbols we can avoid these intermediate considerations. But to the author's knowledge a theory of modular symbols is only available for elliptic and Hilbert modular forms.

We conclude that a software layer that encodes modular forms as abstract elements is an essential feature of any implementation dealing with their Fourier expansions. For this reason, we introduce graded rings (or modules) of expansions.

**Definition 3.23** *Fix a* $\mathbb{G}\mathrm{r}$*-grading of* $A\{x_1, \dots, x_n\}$ *or* $A\langle x_1, \dots, x_n\rangle$*.*

*A graded ring of expansions is a morphism*

$$\phi : A\{x_1, \dots, x_n\} \to A[\![\mathscr{F}]\!]_C^G,$$

*such that* $\ker \phi \subseteq \mathrm{gr}^{-1}(\{1\})$.

*A graded module of expansions is a morphism*

$$\phi : A\langle x_1, \dots, x_n\rangle \to A[\![\mathscr{F}]\!]_C^G,$$

*such that* $\ker \phi \subseteq \mathrm{gr}^{-1}(\{1\})$.

We call the image of an element under the morphism of the preceding definition its (Fourier) expansion.

Usually, rings or modules of modular forms for a fixed group do not have finite rank over their base ring $R$. Only if we fix a cocycle, that is, if we fix a weight, the according module will have finite rank. Scalar cocycles form a group and noncommutative cocycles are acted on by them. In most cases the grading that we impose will coincide with either the group of scalar cocycles or the set of all cocycles. Besides the theoretical meaning of cocycles, the modules of modular forms of fixed weight have finite rank. This allows for comparison of modular forms by checking finitely many Fourier coefficients; this is a key feature of modular forms, that plays an important role in many applications.

## 4. The framework

In this section we briefly outline the framework that implements the ideas presented in Section 3. This framework is available on the author's homepage, where the reader will also find instructions for installing it. The documentation contained in the implementation's docstrings, that can be found in the folder `fourier_expansion_framework` gives more detailed information about its behavior. An example implementation, illustrating how to implement the most important methods, is provided in the folder `algebraicpowerseries`. This example comes with documentation that instructs the reader how to implement his own rings of Fourier expansions.

We adapt the notation from the preceding section. For (nonequivariant) monoid powers series we assume that $\mathscr{F} : \underline{\Lambda} \to \underline{\mathrm{Abs}}_{\mathrm{cofinite}}(S)$ maps to the category of *cofinite* subsets of $S$.

### *4.1. Multiplication of monoid power series*

Multiplication of monoid power series is implemented naively. Many monoids occurring in applications can be embedded into $\mathbb{N}^n$ for some $n$. Consequently, fast multiplication using Karatsuba multiplication (see [12, 30]) is available. In practice, though, the increased memory consumption makes this approach useless for equivariant monoid power series. Nevertheless, we can improve the naive multiplication for equivariant monoid power series. For every $G$-orbit $Gs \subseteq S$ we only have to calculate the sum $\sum_{s_1 s_2 = s} a_{s_1} b_{s_2}$ for one representative $s$. Implementing multiplication in $A[\![\mathscr{F}]\!]$ over a monoid $S$ requires the programmer only provide a function computing $\{(s_1, s_2) \,:\, s_1 s_2 = s\} \subset S^2$ for any given $s \in S$. Alternatively, a function computing these coefficients given $s$ and $\{(s, a_s)\}$ and $\{(s, b_s)\}$ can be provided. The latter approach can be taken, when it is particularly important to have fast multiplication available.

### *4.2. Reduction of indices*

We will call $s \in S$ an index of the expansion $\sum a_s s \in A[\![\mathscr{F}]\!]^G$. The implementation will store coefficients only for special elements in each $G$-orbit. We call these elements reduced indices. We do not require that every orbit contains only a single reduced index. The more indices are considered reduced, the more memory an equivariant monoid power series will consume. The programmer is required to provide a function that associates to each index a reduced index in the same orbit.

Recall that the transition from $\mathscr{F}$ to $\mathscr{F}^G$ is performed automatically. This results in the following restriction on reduced indices $s$: For every $\lambda \in \mathrm{Ob}(\underline{\Lambda})$ with $s \in \mathscr{F}(\lambda)$ we require $s \in \mathscr{F}^G(\lambda)$. In other words, the $G$-symmetrization $\mathscr{F}^G(\lambda)$ of $\mathscr{F}(\lambda)$ must contain all reduced indices contained in $\mathscr{F}(\lambda)$.

### *4.3. Filters and precisions of monoid power series*

The epimorphisms $\epsilon_\lambda$ attached to $\mathscr{F}$ are implemented using the notion of filters. Within the framework, $\lambda \in \mathrm{Ob}(\underline{\Lambda})$ corresponds to a filter $f_\lambda \subseteq S$. By definition, $s \in f_\lambda$ if and all if $s \notin \mathscr{F}(\lambda)$. We agree on this to conform to the user's intuition and well established conventions in Sage.

An element $a \in A[\![\mathscr{F}]\!]^G$ that is approximated by $\epsilon_\lambda(a) \in \mathscr{F}^{\mathrm{Quot}(A)}(\lambda)$ is said to have precision $\lambda$ or $f_\lambda$. Such an equivariant monoid power series will be stored by saving the set $\{(s, a_s)\}$ where $s$ runs through the set of reduced indices contained in $f_\lambda$.

In addition to filters $f_\lambda$, that stem from $G$-cofinite sets $\mathscr{F}(\lambda)$, there are also filters implemented that are the union of infinitely many distinct $G$-orbits. Obviously, the computer cannot store infinitely many coefficients. Instead, an element of $A[\![\mathscr{F}]\!]^G$ with such a precision $f$ attached to is understood to have almost only vanishing coefficients for indices in $f$. Every implementation must provide a filter $f_S = S$, that is used to store elements of $A$.

### *4.4. Types of modular forms*

In Section 3 we have discussed rings of Fourier expansions and rings that wrap them, that we called rings of graded expansions. The relation between both is encoded in *types of modular forms*:

$$\overbrace{\text{equivariant monoid power series} \overset{\text{conversion}}{\dashleftarrow\!\dashrightarrow} \text{graded expansions}}^{\text{types of modular forms}}.$$

The programmer must provide types of modular forms, that the framework will use to automatically initialize all other classes. We content ourselves with giving a list of supported features:

- Generators with Fourier expansion and weight,
- relation between generators,
- transparent and algebraically correct base changes,
- attached Hecke operators and
- associated notions of vector-valued and scalar-valued modular forms.

For a detailed and guiding documentation, the reader is referred to the file `howtoimplementamodularform` \`example_type.py`.

## 5. Usage

In this section, we illustrate how to use the framework that we have described in the preceding section. It is based on an implementation of Siegel modular forms. This implementation was initiated by N. Ryan and N. Skoruppa and later continued by a larger group, of which the author is member [26]. It was then ported to this framework by the author, resulting in higher performance and greater flexibility. Whenever we refer to "this implementation" we mean the implementation [23], that is available on the author homepage.

To make the reader familiar with the basic commands and ideas of the implementation we choose a twofold approach. Section 5.1 and 5.2 contain two examples, that illustrate how quick and easy calculations can be using the implementation. In Section 5.3 and 5.4 we give more systematic and more detailed instructions for the user.

Notice that neither of the examples in the first two sections is new. We have chosen them, because they do not require deep knowledge about Siegel modular forms and because they can be run on every laptop. The code for all examples presented in this section, with many comments included, can be found in the folder `howtoimplementamodularform`. The reader is encouraged to carefully inspect the code and to get familiar with basic operations of the implementation and, if necessary, of Sage by performing similar calculations. All code was written assuming that the framework has been installed according to the instructions in `README`.

## 5.1. Siegel modular forms and Hecke actions

We will demonstrate that the space of weight 50 Siegel modular forms of degree 2 decomposes into 4 irreducible, simple Hecke modules over $\mathbb{Q}$. This kind of computation was first done in [31] almost 20 years ago. But it has not been pursued since. Only recently, the author studied the rational Hecke action on spaces of Siegel modular forms up to weight 150 using this implementation (see [24]).

Comments, that are not given in the following listing, can be found in the sage-files.

```
from paramodularforms import *

SR = SiegelModularFormsG2(QQ,
         SiegelModularFormG2_Classical_Gamma(), 400)
sm = SR.graded_submodule(50)
sm._check_precision() ## Result: True
sm.rank() ## Result: 31

hecke_hom = sm.hecke_homomorphism(2)
minpol = hecke_hom.minpoly()
minpol_fac = minpol.factor()
[(p.degree(), e) for (p,e) in minpol_fac]
## Result: [(1, 1), (3, 1), (7, 1), (20, 1)]
```

Since the degrees of all Hecke components with respect to 2 sum up to the rank of the module $1 + 3 + 7 + 20 = 31$, the claim is proved.

Let us say a few words about the code. First of all, we need to import the Python module that provides the implementation of Siegel modular forms. It is called `paramodularforms`, since it also contains an implementation of paramodular forms, that Siegel modular forms form a special case of.

Notice that in this implementation the symmetric matrices $s = \left(\begin{smallmatrix} a & b/2 \\ b/2 & c \end{smallmatrix}\right)$ that we used in Example 3.4 correspond to triples $(a, b, c)$.

The precision of a Fourier expansion for Siegel modular forms of degree 2 is expressed in terms of discriminant filters by default. That is, a positive definite index $(a, b, c)$ is contained in a filter with index $D$ if and only if $4ac - b^2 < D$.

To compute the matrix of the Hecke operator $T(2)$ with respect to a fixed basis of Siegel modular forms it is applied to the Fourier expansions of these forms. Then the framework will try to express the resulting Fourier expansions uniquely in terms of the Fourier expansions of the basis. Care must be taken of the precision. Applying the Hecke operator $T(n)$ to a Siegel modular form of degree 2 will reduce its precision by a factor $n^{-2}$. Consequently, although all Siegel modular forms of weight 50 are uniquely determined by their expansion with precision 100, we initialize the ring with precision 400.

## *5.2. An extremal lattice and the attached Siegel theta series*

We compute the Fourier expansion of the Siegel modular theta series attached to the extremal, unimodular lattice of dimension 72 that was constructed in [19]. In particular, we check the number of minimal vector in this lattice. More precisely, we check, assuming that there is an even, unimodular lattice $L$ of dimension 72, that the minimal length of $L$ is 8 and the number of minimal vectors in $L$ is 6218175600. Both claims have been proved recently in [19].

```
from paramodularforms import *

SR = SiegelModularFormsG2(QQ,
        SiegelModularFormG2_Classical_Gamma(), 100)
sm = SR.graded_submodule(36)
sm._check_precision() # Result: True
sm.rank() # Result: 17

fe_indices = filter( lambda (a,b,c): a < 4 and c < 4,
                     SR.fourier_expansion_precision() )
fe_indices.remove((0,0,0))
fe_basis = [b.fourier_expansion() for b in sm.basis()]
relations = matrix(QQ, [ [ b[i] for i in fe_indices ]
                          for b in fe_basis ])
ker = relations.left_kernel()
ker.rank() # Result: 1
extremal_form = SR( sum( [b * c
      for (b,c) in zip(sm.basis(), ker.basis()[0])] ) )
extremal_form = extremal_form \
  * extremal_form.fourier_expansion()[(0,0,0)]**-1

extremal_count = \
  sum( extremal_form.fourier_expansion()[(4,b,4)]
         for b in range(-8, 9) )
sqrt(extremal_count) # Result: 6218175600
```

This proves the claim, since all pairs $(v_1, v_2)$ of minimal vectors in $L$ result in a Fourier index $[4, b, 4]$ for some $b$. We remark that the interested reader may check whether the Fourier coefficients of `extremal_form` are indeed integral and positive.

All comments on this piece of code, that are quite technical, can be found within the code.

## *5.3. A brief tour of Fourier expansions*

### *5.3.1. Precisions and basic arithmetic*

This section contains a description of the most basic methods of (Fourier) expansions provided by the framework. To keep everything as simple as possible we use power series, that we have described in Example 3.3. In particular, we will be concerned with *non*equivariant monoid power series. Notice that all methods that we introduce are also provided for equivariant monoid power series.

We start with importing the power series module. Again we assume that the framework has been installed according to the instructions in `README`.

```
sage: from algebraicpowerseries import *
```

With this module imported we can create a ring of power series and define some elements:

```
sage: R = PowerSeriesRing_mult(ZZ, 3)
sage: x,y,z = R.gens()
sage: f = x + y + z
sage: g = x + y
sage: h = x**10
```

The coefficients of monoid power are usually not printed, since in most situations there are too many. Normally, the user wants to access single coefficients. This can be done using the usual bracket notation:

```
sage: g[(0,1,0)] # coefficient of y
1
sage: g[(0,1,2)] # coefficient of y z^2
0
```

A dictionary with all nonvanishing coefficients (and possibly more) can be obtained as follows:

```
sage: g.coefficients()
{(1, 0, 0): 1, (0, 1, 0): 1}
sage: h.coefficients()
{(10, 0, 0): 1}
```

Arithmetic operations are performed as usual.

```
sage: (h + g)[(10,0,0)]
1
sage: (h * h).coefficients()
{(20, 0, 0): 1}
```

Note that `f`, `g` and `h` have infinite precision. This makes them no different from multivariate polynomials, that are already provided in Sage with vastly faster multiplication. The next step is hence to discuss filters. Filters for multivariate power series are implemented in the class `NNnFilter`. The user may assign them to `h` as follows:

```
sage: f.precision()
Filtered NN^3 up to (+Infinity, +Infinity, +Infinity)
sage: ht = h.truncate(NNnFilter((3,3,3)))
sage: ht.precision()
Filtered NN^3 up to (3, 3, 3)
```

The filer $(3, 3, 3)$, that we have attached to `ht`, means that only monomials $x^{e_x} y^{e_y} z^{e_z}$ with $e_x, e_y, e_z < 3$ have known coefficients. Consequently, it is not possible to query the following coefficient:

```
sage: ht[(10,0,0)]
ValueError: (10, 0, 0) out of bound
```

*5.3.2. Modules spanned by expansions*

The arithmetic of monoid power series is rather restricted, since they are quite general objects, but linear operations are fully supported. Modules spanned by expansions are the most important tool when computing with modular forms. A special implementation may be initialized as follows:

```
sage: from fourier_expansion_framework import *
sage: em = ExpansionModule([f,g])
sage: em.dimension()
2
```

Expansion modules have an abstract basis attached to, but their purpose is to provide access to the underlying expansions. The next statement checks whether the basis's expansions are linearly dependent and whether these expansions truncated to $xyz$, which corresponds to applying the filter $(1, 1, 1)$, are linearly independent.

```
sage: em._check_precision()
True
sage: em._check_precision(NNnFilter((1,1,1)))
False
```

This is the case for the first example, because `f` and `g`, that are linearly independent, have infinite precision. But truncating them to $xyz$ yields zero for both of them, resulting in a linear dependence. We remark that if the basis elements have different precisions attached to, the minimum of all of them is used when comparing the basis elements. Recall from Section 4 that $\underline{\Lambda}$ is a net. Hence, such a filter always exists.

To understand what a basis or rather *abstract basis* is to an expansion module we have a look at the following lines of code.

```
sage: em = ExpansionModule([f,f,g])
sage: em.dimension()
3
sage: em._check_precision()
False
```

The first and second copy of `f` are considered different elements of the (abstract) basis. But they have the same expansions attached to, and hence they are linearly depended even though they have infinite precision. Nevertheless, by choosing an appropriate set of elements, we can span a submodule every element of which has an expansion attached to that is a unique linear combination of the basis's expansions. To obtain a minimal subset of generators that spans the space of attached expansions the user may call the following method:

```
sage: em.pivot_elements()
[(1, 0, 0), (0, 0, 1)]
```

The main application of this method is to analyze spaces of Fourier expansions of modular forms that have insufficient precision.

It is often useful to consider an expansion module as a homomorphism from the free module over the abstract basis to another free module over the base ring. This homomorphism can be obtained as follows:

```
sage: em = ExpansionModule([f,g])
sage: em.fourier_expansion_homomorphism()
Free module morphism defined by the matrix
(not printing 2 x 8 matrix)
Domain: Module of Fourier expansions in ...
Codomain: Ambient free module of rank 8 over ...
```

This homomorphism – call it $\phi$ – provides us with a possibility to describe the pivot elements in more detail. They form a set of representatives of a basis of the coimage of $\phi$. To get hands on it, we compute the kernel of $\phi$:

```
sage: em.fourier_expansion_kernel()
Free module of degree 2 and rank 0 over Integer Ring
...
sage: em = ExpansionModule([f,f,g])
sage: em.fourier_expansion_kernel()
Free module of degree 3 and rank 1 over Integer Ring
Echelon basis matrix:
[ 1 -1  0]
```

*5.3.3. Converting abstract elements and expansions back and forth*

It is easy to obtain the expansion attached to any element of an expansion module:

```
sage: em.basis()[0].fourier_expansion()
Monoid power series in Ring of monoid power series ...
```

It is more involved to express an expansion in terms of the basis elements. Recall that `f` occurs twice in the basis of `em`.

```
sage: em.coordinates(2 * g + f)
ValueError: Insufficient precision of submodule ...
sage: em.coordinates(2 * g + f, force_ambiguous = True)
[1, 0, 2]
```

The first try to obtain coordinates fails, since $2g + f$ cannot be expressed unambiguously. The additional keyword in the second command instructs the framework to ignore this issue.

Notice that the error raised by the framework indicates that the expansion module does not have sufficient precision. The next example illustrates the case when the module has sufficient precision, but the expansion that we intend to convert does not.

```
sage: em = ExpansionModule([f,g])
sage: em.coordinates(f.truncate(NNnFilter((1,1,1))))
ValueError: No unambiguous coordinates available
```

Every expansion module has a base ring attached to it. In general, it will be the minimal ring that contains all base rings of the basis's expansions. The base ring being fixed can result in issues, if a potential result is only contained in a base extension. We give two examples of this problem. The first requires a base change from $\mathbb{Z}$ to $\mathbb{Q}$, and the second requires a base change to the cyclotomic field $\mathbb{Q}(\zeta_3)$.

```
sage: em.base_ring()
Integer Ring
sage: em.coordinates(g / 2 + f)
ArithmeticError: No coordinates ...
sage: em.coordinates(g / 2 + f, in_base_ring = False)
(1, 1/2)
sage: K.<rho> = CyclotomicField(3)
sage: em.coordinates(g * rho, in_base_ring = False)
(0, rho)
```

*5.4. A brief tour of rings of modular forms*

In this section we revisit the ring of Siegel modular forms, that we have used in Section 5.1 and Section 5.2. The aim of this section is to give a more complete overview of the provided features. We start with initializing the ring of modular forms as before.

The function `ModularFormsAmbient` may be used to initialize any kind of module or ring of modular forms. The syntax is as follows: The first argument is the base ring and the second argument is a type of modular forms, that we have described in Section 4.4. The third argument is a filter or any value that can be converted to a filter by the type. Usually, this will result in a canonical filter. In our case the third argument is the integer 64, resulting in a filter that contains all quadratic forms with negative discriminant less than 64.

```
sage: from fourier_expansion_framework import *
sage: from paramodularforms import *
sage: SR = ModularFormsAmbient(QQ,
           SiegelModularFormG2_Classical_Gamma(), 64)
```

The generators of Siegel modular forms were given by Igusa [13]. They are called `I4`, `I6`, `I10` and `I12`. The Fourier expansion of the generators will be equivariant power series. Recall from Section 5.2 that the indices are triples $(a, b, c)$, that correspond to symmetric matrices $\left(\begin{smallmatrix} a & b/2 \\ b/2 & c \end{smallmatrix}\right)$.

```
sage: fe = SR.3.fourier_expansion()
sage: fe[(1,0,3)]
736
```

The precision of the Fourier expansion of generic elements is 64, as we expect. The precision of constants is nonetheless infinite:

```
sage: fe.precision()
Discriminant filter (64)
sage: fe.parent().zero().precision()
Discriminant filter (+Infinity)
```

We calculate the eigenforms with respect to the Hecke operator $T(2)$. Notice that for each Galois orbit of eigenforms over the base ring $\mathbb{Q}$ the framework returns exactly one representative.

```
sage: sm = SR.graded_submodule(26)
sage: hefs = sm.hecke_eigenforms(2)
sage: len(hefs)
5
sage: map(lambda f: f.is_cusp_form(), hefs)
[False, True, True, False, True]
```

Next, we consider the Maaß spezialschar in `sm`. Its rank is 4, and exactly one basis element is not a cusp form.

```
sage: mm = sm.maass_space()
sage: mm.rank()
4
sage: mm.ambient_module() == sm
True
sage: mm.0
```

```
(1/1728, 0, 33125/84873096, 0, 0, ...
sage: SR(mm.0).is_maass_form()
True
sage: SR(mm.0).is_cusp_form()
False
sage: SR(mm.1).is_cusp_form()
True
sage: (SR.1*SR.2^2).is_maass_form()
False
```

The framework allows for adding forms with distinct weights. The result will not have any invariance with respect to the modular group, but we are able to extract the homogeneous components. Notice, that only homogeneous element can be Maaß forms. In particular, sums of Maaß forms with distinct weights are not considered Maaß forms.

```
sage: a = SR.0 + SR.1
sage: a.homogeneous_components()
{4: I4, 6: I6}
sage: a.is_maass_form()
False
```

## 6. Available packages

This section contains a brief list of packages that come with the framework. We do not give many details, but rather restrict to outlining features that are currently implemented. The usage is the same as for the Siegel modular forms implementation, that we used in the preceding section. For details we refer the interested reader to the source code and its documentation.

A) **Scalar-valued Siegel modular forms** *Included in the Python module paramodularforms*

   The implementation of Siegel modular forms currently provides generators for the ring of even weight forms for the full modular group. The Maaß spezialschar is implemented. The user can construct arbitrary Maaß lifts from a pair of elliptic modular forms in $M_k \times S_{k+2}$. In contrast, it is not possible to use Jacobi forms as input. The user can test whether forms are cuspidal or not. The multiplication of forms is implemented in Cython and is thus fast.

   There is also a special method for spanning spaces of fixed weight by products of at most two elements of the Maaß spezialschar. Moreover, all Hecke operators are available.

B) **Vector-valued Siegel modular forms** *Included in the Python module paramodularforms*

Satoh brackets (see [28]) are available for arbitrary scalar-valued Siegel modular forms. Generators for the space of vector-valued weight 2 forms as a module over the ring of scalar-valued forms are provided.

C) **Paramodular forms** *Included in the Python module paramodularforms*

All spaces of paramodular forms of prime level which are spanned by products of Gritsenko lifts for the full paramodular group and by symmetrizations of Siegel modular forms can be obtained. Notice that there are spaces which are not spanned by such forms (see [21]), but the first known examples appear for large $p$. The user can construct Gritsenko lifts using Jacobi forms. Also symmetrizations of Siegel modular forms for the full modular group are available.

Hecke operators are available over all good primes. Moreover, Schmidt's operator $T_5$ and the Atkin-Lehner involution (see [29]) can be computed numerically.

D) **Hermitian modular forms** *Included in the Python module hermitianmodularforms*

The ring of symmetric modular forms of even weight for the full modular group over $\mathbb{Q}(\sqrt{-3})$ is implemented using generators that can be found in [8]. This implementation was a project joint with Dominic Gehre. The generators are computed using Borcherd's additive lift (see [3]). Computing their Fourier expansion involves the computation of elliptic Eisenstein series for $\Gamma_1(36)$. This can result in performance issues depending on Sage's performance for elliptic modular forms.

The user can compute the additive lift over the Hermitian modular group $\Gamma^{(2)}(\mathcal{O}_{\mathbb{Q}(\sqrt{-3})})$ for arbitrary input. There is ongoing work to implement a new and fast algorithm for computing the multiplicative lift.

E) **Quaternion modular forms** *Currently not included*

There is an ongoing student project supervised by the author that aims at implementing quaternion modular forms over the Hurwitz order (see [16]).

F) **Jacobi forms** *Included in the Python module jacobiforms*

Jacobi forms of even weight and prime index for the full Jacobi modular group are implemented. They mostly serve as input for Gritsenko lifts, and thus have very limited functionality. The theta decomposition and the Taylor expansion are used to compute the coefficients. This allows for computing expansions up to very high precisions. This results in a drawback of performance when computing Jacobi forms of moderate or high index.

G) **Multivariate algebraic power series** *Included in the Python module algebraicpowerseries*

Algebraic power series are implemented with naive multiplication for the purpose of demonstration.

## 7. Perspectives and related open problems

### *7.1. Tasks to be taken to improve the framework*

In Section 6, we have seen that there is already a multitude of modular forms implemented using this framework. Needless to say that there remains a lot of work to be done. In particular, Hecke operators are implemented only for scalar-valued paramodular forms. This, by far, does not correspond to their significance.

Certainly, most work can be done by providing further implementations. The implementation of Siegel modular forms demonstrates to what great extent a generic implementation can be useful. In Section 6, we have seen for which types of modular forms initial work has been done. Each of them, though, suggests various improvements, ranging from congruence subgroups, that may be implemented, to Hecke operators and the attached $L$-series, that have only been considered in view cases.

The framework in general, we conclude, is in good shape, and it is ready for future extensions. It is unsatisfactory, however, that only naive multiplication is currently used throughout the framework. A Karatsuba type multiplication for equivariant monoid power series is not in sight. Research in this direction will focus on specific monoids that we understand particularly well. Siegel modular forms of degree 2 can profit from this, and the framework is ready to support new multiplication algorithms as soon as they are available.

### *7.2. Potential research applications based on the framework*

There are many applications of elliptic modular forms relating their coefficients to interesting quantities, examples of which are Hurwitz class numbers (see [6]), partitions (see [2]) and traces of singular moduli (see [38]). In contrast, relatively few applications of coefficients of higher degree modular forms have been found until today. In this section we suggest three applications, that can be treated by means of this framework.

A) Fix $N \in \mathbb{N}$. Denote the space of Gritsenko lifts to weight $k$ and level $N$ paramodular forms by $G_k^{[N]}$. We write $\mathrm{Sym}_k$ for the symmetrization of Siegel modular forms of weight $k$ for the full Siegel modular group to the full paramodular group of level $N$. The space of weight $k$ and level $N$ paramodular forms is denoted by $\mathbb{M}_k^{[N]}$. In [24] the author proved that for $k \le 172$

$$\mathbb{M}_k^{[1]} = \bigoplus_{k' \le k} G_{k'}^{[1]} G_{k-k'}^{[1]}.$$

He also conjectured that the equality above holds for all $k$. A similar result possibly holds for paramodular forms, if $k$ is sufficiently large:

$$\mathbb{M}_k^{[N]} = \bigoplus_{k' \le k} G_{k'}^{[N]} G_{k-k'}^{[N]} \oplus \bigoplus_{N' | N} \mathrm{Sym}_k^N \mathbb{M}_k^{[N']}.$$

This has only been checked in few cases. We remark that for sufficiently high level $N$ there are weight 2 forms that are not Gritsenko lifts (see [21]). Consequently, the conjecture cannot possibly hold for all $k$. Gritsenko suggested that lifts of weight 1 Jacobi forms with character might close this gap, but no effort has been made to study these lifts in the context of this conjecture.

There is a deep motivation behind this question: It is a classical theorem that products of at most two Eisenstein series span the space of elliptic modular forms. Even the (linear) relations can be described. The proof is outlined in [15], and it is based on periods. Presumably, a proof of the conjecture above will lead to new insight into the theory of periods of special Siegel modular forms.

B) The vanishing cone for paramodular forms was investigated in [21, Sec 5]. It is directly related to pivot sets, that have recently been investigated by the author in [24, Sec 4]. One is interested in certain maps $\phi$ from the set of Fourier indices of Siegel modular forms, that is, even quadratic forms, to $\mathbb{R}_+$. Fix a weight $k$ form $f$. The support of $f$, that is, the set of indices with nonvanishing coefficients, is denoted by $\mathrm{supp}(f)$. We aim at proving Sturm bound like statements

$$\phi(\mathrm{supp}(f)) \subseteq [c, \infty) \Rightarrow f = 0$$

with optimal $c = c_{\mathrm{anal}} k$. In [24] the author conjectured that for the choice $\phi(t) = \sqrt{|\operatorname{disc} t|}$ the constant $c$ given in [21] can be improved asymptotically. Evidence was provided based on computations, that were done with this framework. Analogously, considerations for $\phi$ the dyadic trace or the trace can be done. Investigating the vanishing cone itself would be of greatest use for future applications.

A related, more refined question focuses on arithmetic vanishing. For an integral Siegel modular form we denote by $\mathrm{supp}_p(f)$ the set of indices with coefficients not divisible by a prime $p$. We aim at finding minimal $c = c_{\mathrm{arith}} k$ such that

$$\forall p \text{ prime} : \big(\phi(\mathrm{supp}_p(f)) \subseteq [c, \infty) \Rightarrow f \equiv 0 \,(\mathrm{mod}\, p)\big).$$

This kind of statement can be considered an arithmetic analog of the analytic statements above. The author proved that for all weights $k \le 80$ the arithmetic and analytic vanishing bounds $c_{\mathrm{arith}}$ and $c_{\mathrm{anal}}$ for $\phi(t) = \sqrt{|\operatorname{disc} t|}$ coincide. This answers a question raised by Poor and Yuen in private correspondence. Notice that for elliptic modular forms for the full modular group the Victor-Miller basis implies the equality of arithmetic and analytic vanishing bounds.

Until now considerations focused on $\phi(t) = \sqrt{|\operatorname{disc} t|}$, although the question can be formulated for a whole class of functions $\phi$. In particular, it is not clear whether arithmetic and analytic vanishing bounds will differ for any $\phi$.

C) Expressing particular modular forms in terms of theta series or Maaß lifts is an interesting challenge. Progress in this area can reveal deep properties of modular forms such as positivity of Fourier coefficients.

Let $\Delta_{30}$ be the Siegel modular form of weight 30 for the full Siegel modular group with character, that is unique up to scalar multiplicities. The weight 30 Hermitian modular form $\phi_{30}$ for the full Hermitian modular group over $\mathbb{Q}(\sqrt{-1})$ is also unique up to scalar multiplicities (see [8]). In [11] Gehre and Krieg gave the following result: Let $\Theta_1, \ldots, \Theta_6$ be the six quaternion theta series defined in [9]. We denote the restriction to the Siegel upper half space and the Hermitian upper half space by $\cdot|_{\mathbb{H}_\mathrm{S}}$ and $\cdot|_{\mathbb{H}_\mathrm{H}}$. With this notation we have

$$\Delta_{30} = F\big|_{\mathbb{H}_\mathrm{S}} \quad \text{and} \quad \phi_{30} = F\big|_{\mathbb{H}_\mathrm{H}},$$

where

$$\begin{aligned} F &:= (\Theta_5 + \Theta_6)(\Theta_2^2 - \Theta_3^2)(\Theta_2^2 - \Theta_4^2)(\Theta_3^2 - \Theta_4^2) \\ &\qquad \prod_{\substack{(\epsilon_1,\ldots,\epsilon_4)\in\{\pm1\}^4 \\ \epsilon_1,\cdots,\epsilon_4=-1}} \big(\epsilon_1\Theta_1 + \epsilon_2\Theta_2 + \epsilon_3\Theta_3 + \epsilon_4\Theta_4 + \Theta_5 + \Theta_6\big) \end{aligned}$$

These results were proved in a purely computational way, and it is likely that further useful results can be obtained using the author's framework.

## References

[1] Alfes, C., Jameson, M., Oliver, R. L., 2011. Proof of the Andrews-Alder Conjecture. Proc. of the Amer. Math. Soc. 139 (1), 63–78.

[2] Andrews, G. E., 1976. The theory of partitions. Addison-Wesley Publishing Co., Reading, Mass.-London-Amsterdam, encyclopedia of Mathematics and its Applications, Vol. 2.

[3] Borcherds, R. E., 1998. Automorphic forms with singularities on Grassmannians. Invent. Math. 132 (3), 491–562.

[4] Bosma, W., Cannon, J., Playoust, C., 1997. The Magma algebra system. I. The user language. J. Symbolic Comput. 24 (3-4), 235–265.

[5] Bringmann, K., Richter, O., Raum, M., 2011. Kohnen's limit process. In preparation.

[6] Bruinier, J. H., van der Geer, G., Harder, G., Zagier, D., 2008. The 1-2-3 of modular forms. Universitext. Springer-Verlag, Berlin.

[7] Conley, C., Raum, M., 2010. Harmonic maaß-jacobi forms of degree 1 with higher rank indices. arXiv:1012.2897 [math.NT].

[8] Dern, T., Krieg, A., 2003. Graded rings of Hermitian modular forms of degree 2. Manuscripta Math. 110 (2), 251–272.

[9] Freitag, E., Hermann, C. F., 2000. Some modular varieties of low dimension. Adv. Math. 152 (2), 203–287.

[10] Gehre, D., Kreuzer, J., Raum, M., 2011. Hermitian borcherds products. In preparation.

[11] Gehre, D., Krieg, A., 2010. Quaternionic theta constants. Arch. Math. 94, 59–66.

[12] Hart, B., et al., 2010. Fast Library for Number Theory 1.5.2. `http://www.flintlib.org`.

[13] Igusa, J.-i., 1962. On Siegel modular forms of genus two. Amer. J. Math. 84, 175–200.

[14] Kohnen, W., Kuß, M., 2002. Some numerical computations concerning spinor zeta functions in genus 2 at the central point. Math. Comp. 71 (240), 1597–1607.

[15] Kohnen, W., Zagier, D., 1984. Modular forms with rational periods. In: Modular forms (Durham, 1983). Ellis Horwood Ser. Math. Appl.: Statist. Oper. Res. Horwood, Chichester, pp. 197–249.

[16] Krieg, A., 1985. Modular forms on half-spaces of quaternions. Vol. 1143 of Lecture Notes in Mathematics. Springer-Verlag, Berlin.

[17] Krieg, A., Raum, M., 2009. The functional equation for the twisted spinor-l-function of genus 2. arXiv:0907.2767 [math.NT].

[18] Mac Lane, S., 1998. Categories for the working mathematician, 2nd Edition. Vol. 5 of Graduate Texts in Mathematics. Springer-Verlag, New York.

[19] Nebe, G., 2010. An even unimodular 72-dimensional lattice of minimum 8. arXiv:1008.2862v3 [math.NT].

[20] Poor, C., Yuen, D. S., 2007. Computations of spaces of Siegel modular cusp forms. J. Math. Soc. Japan 59 (1), 185–222.

[21] Poor, C., Yuen, D. S., 2009. Paramodular cups forms. arXiv:0912.0049v1 [math.NT].

[22] Raum, M., 2009. Elementary divisor theory for the modular group over quadratic field extensions and quaternion algebras. Submitted for publication.

[23] Raum, M., 2010. Modular forms framework, `http://people.mpim-bonn.mpg.de/mraum/en/downloads.xml`.

[24] Raum, M., 2011. Efficiently generated spaces of classical Siegel modular forms and the Böcherer conjecture. to appear in J. Aust. Math. Soc.

[25] Raum, M., 2011. Hecke algebras related to the unimodular and modular groups over quadratic field extensions and quaternion algebras. Proc. Amer. Math. Soc. 139, 1321–1331.

[26] Raum, M., Ryan, N. C., Skoruppa, N.-P., Tornaría, G., 2009. Siegel modular forms package, `http://hg.countnumber.de`.

[27] Raum, M., Ryan, N. C., Skoruppa, N.-P., Tornaría, G., 2011. Theoretical and Algorithmic Aspects of an Implementation of Siegel Modular Forms. In preparation.

[28] Satoh, T., 1986. On certain vector valued Siegel modular forms of degree two. Math. Ann. 274 (2), 335–352.

[29] Schmidt, R., 2005. Iwahori-spherical representations of GSp(4) and Siegel modular forms of degree 2 with square-free level. J. Math. Soc. Japan 57 (1), 259–293.

[30] Shoup, V., et al., 2010. Number Theory Library 5.5.2. `http://www.shoup.net/ntl/`.

[31] Skoruppa, N.-P., 1992. Computations of Siegel modular forms of genus two. Math. Comp. 58 (197), 381–398.

[32] Springer, T. A., 1977. Invariant theory. Lecture Notes in Mathematics, vol. 585. Springer-Verlag, Berlin.

[33] Stein, W. A., et al., 2011. Purple sage. `ttp://purple.sagemath.org/`.

[34] Stein, W. A., et al., 2011. Sage Mathematics Software (Version 4.6.1). The Sage Development Team, `http://www.sagemath.org`.

[35] Stein, W. A., et al., 2011. Sage Tutorial. `http://www.sagemath.org/doc/tutorial/`.

[36] van Rossum, G., 1995. Python tutorial. Tech. Rep. CS-R9526, Centrum voor Wiskunde en Informatica, Amsterdam.

[37] van Rossum, G., et al., 2011. Python documentation. `http://docs.python.org/`.

[38] Zagier, D., 2002. Traces of singular moduli. In: Motives, polylogarithms and Hodge theory, Part I (Irvine, CA, 1998). Vol. 3 of Int. Press Lect. Ser. Int. Press, Somerville, MA, pp. 211–244.

[39] Zimmermann, P., et al., 2010. Calcul mathématique avec Sage. `http://sagebook.gforge.inria.fr/`.